Amendments to the Specification:

Please replace paragraph [0022] with amended paragraph [0022] as follows:

[0022] **FIG. 13** illustrates the use of the present invention in conjunction with a variable length flat ~~pate~~ page table.

Please replace paragraph [0037] with amended paragraph [0037] as follows:

[0037] Several of the improvements accomplished by the present invention can be conceptually illustrated through a comparison of **FIG. 1** and **FIG. 2**. **FIG. 1** represents a typical prior art approach to task scheduling for a coprocessor. A buffer is provided which can be accessed by various applications, *e.g.,* Application 1, Application 2, and Application 3. The applications can load tasks for the coprocessor into a buffer, and those tasks can be processed by the coprocessor after previously submitted tasks are completed. As illustrated, this approach leaves open a potential "hogging" of the coprocessor. In **FIG. 1**, App. 1 is hogging the coprocessor. App. 1 has requested that the coprocessor work on ~~seven~~ eight tasks, while the other two applications combined have requested work on only three tasks. In situations like these where multiple applications need the coprocessor, a system such as that provided by **FIG. 2** may provide improved functionality.

Please replace paragraph [0051] with amended paragraph [0051] as follows:

[0051] Step 6 represents the submission of the command buffer to a ~~kernel mode driver.~~ coprocessor kernel. The coprocessor kernel can direct the command buffer to a kernel mode driver. The kernel mode driver may generally be a driver, as described above with reference to the user mode driver, except that the kernel mode driver can operate in kernel mode, as its name suggests. In this regard, a kernel mode driver can be responsible for translating a command buffer into a DMA buffer. IHVs may consider providing the appropriate mechanisms to ensure proper validation and copying of command buffers into kernel mode allocated DMA buffers. DMA buffers may be hardware-specific, in that they are collections of commands ultimately destined for a coprocessor and therefore should properly interface with the coprocessor and supporting hardware.

Please replace paragraph [0091] with amended paragraph [0091] as follows:

[0091] For example, the following elements of the basic scheduling model described above can be eliminated through the use[[r]] of a per-coprocessor-context address space:

1) Patching command buffers by replacing handles with actual memory locations

2) Validating command buffers for memory access

3) Building memory resource lists in kernel mode

4) Creating separate command and DMA buffers

5) Bringing resources for interrupted DMA buffers back a pre-interruption location

Please replace paragraph [0099] with amended paragraph [0099] as follows:

[0099] Refer to **FIG. 10** and **FIG. 11** for embodiments of scheduling in the basic model and scheduling in the advanced model. As will become clear, the advanced model has two scheduling options. When scheduling without demand faulting, a preparation phase can be implemented. When the advanced model uses demand faulting, however, no preparation phase is necessary.

In addition, ~~**FIGS. 12(A), 12(B),** and **12(C)**~~ **FIG. 12(A) and FIG. 12(B)** provide a flowchart demonstrating pseudocode capable of implementing the advanced scheduling model.

Please replace paragraph [0103] with amended paragraph [0103] as follows:

[0103] **The variable length flat page table.** The use of the present invention in conjunction with a variable length flat ~~pate~~ page table is illustrated in **FIG. 13**. In this method, the address space of the coprocessor is virtualized through the use of a flat page table. The virtual address space can be divided into pages of a predefined memory amount, for example 4KB. For each page in the virtual address space, a page table is provided that contains identifiers, for example 64-bit entries, for specifying a physical address and location (*e.g.,* Accelerated Graphics Port (AGP), Peripheral Component Interconnect (PCI), or Video) of associated physical memory. In one embodiment, the page size supported by the

coprocessor is not arbitrary and must be 4KB in order to allow the coprocessor page table to reference system memory pages. Furthermore in this embodiment, the coprocessor page table must be able to address both local video memory and system memory from the same address space. The coprocessor can require that all pages belonging to a single surface be mapped to a single type of memory. For example, the coprocessor can require that all pages belonging to a particular render target be mapped into local video memory. However, page table entries that map surfaces to a variety of physical memory types (AGP, local video, etc.) can coexist in the page table.

Please replace paragraph [0161] with amended paragraph [0161] as follows:

[0161] The Coprocessor Trace of Scheduling Events. The run list can easily be expanded to a size N when the hardware provides some history information of scheduling events to the scheduler. One problem with a simple interrupt is that multiple interrupts can be squeezed together, and it might not be possible to determine exactly what happened to cause an interrupt. This can be addressed, in conjunction with the methods of this invention, by hardware features. [[By]] More particularly, it can be addressed by implementing hardware that can write a context switch history to a specified system memory location readable by the scheduler. To explain this aspect of the invention, consider the following scenario:

1) The scheduler schedules run list A (1-2-3-4-5).

2) A time quantum expires for context #1, and the scheduler sends a new run list B (2-3-4-5-1).

3) While processing the quantum expiration on the CPU, the coprocessor finished with context #1 because it became empty and therefore transitioned to context #2. The coprocessor generated a context switch interrupt for this event.

4) The coprocessor received the notification from the CPU about the new run list, and therefore transitioned to it. The coprocessor generated a context switch interrupt for this event.

5) While processing rendering commands in context #2 of the new run list, the coprocessor encountered a page fault and therefore switched to context #3. The coprocessor generated a context switch interrupt for this event.

6) Context #3 hit a page fault right away and therefore the coprocessor switched to context #4. The coprocessor generated a context switch interrupt for this event.

7) The CPU is finally interrupted for a context switch. Four context switches have actually happened since the original interrupt was raised.

Please replace paragraph [0165] with amended paragraph [0165] as follows:

[0165] With the introduction of memory protection in the advanced scheduling model, DMA buffers sent to the coprocessor may be mostly built by the user-mode driver inside the process of the running application. Those DMA buffers may be mapped in the process of the application, the user-mode driver can write directly to them, and the kernel driver cannot be validating them. DMA buffers might be scribbled on by an application accidentally accessing their virtual addresses or on purpose by a malicious application. In order to allow the driver model to remain secure, *i.e.*, not allow an application to have access to resources it shouldn't have, DMA buffers built in user mode can be limited in what they are allowed to do. In particular, DMA buffers built in <u>user mode</u> can have limited functionality in the following exemplary ways:

1) They can contain only references to virtual address, no reference to physical address at all (including fences).

2) They can not be allowed to contain instructions that would affect the current display (for example CRT, Discretionary Access Control (DAC), Technical Document Management System (TDMS), Television-Out Port (TV-OUT), ~~Internet2~~ <u>Inter Integrated Circuit</u> (I2C) bus).

3) They can not contain instructions that would affect the adapter in general (for example Phase-Locked Loop (PLL).

4) They can have limited power management and/or config space.

5) They can not be allowed to contain instructions that will prevent context switching.

Please replace paragraph [0173] with amended paragraph [0173] as follows:

[0173] Privileged DMA buffers can contain any of the instructions found in a non-privileged DMA buffer. Various preferred embodiments of the invention may implement privileged DMA buffers that ~~allow~~ at least allow the following (explained in further detail in later sections):

1) Insertion of privilege fences

2) Insertion of flip instructions

3) Insertion of "no context-switch" regions

Please replace paragraph [0201] with amended paragraph [0201] as follows:

[0201] What this means is that in double buffering, the scheduler may allow the application to queue one flip and let it continue preparing the DMA buffer for the following frame while the coprocessor finishes rendering the current frame and processes/acknowledges that flip. It also means that [[if]] by the time the application is finished with the preparation of the DMA buffer for the following frame and submits a second flip, it can be blocked until the first flip is acknowledged by the coprocessor.